Exhibit A **BEST AVAILABLE COPY**

D:\q71_jan15_b\quartus\fitter\fsyn\fsyn_atom_dup.h 1

```
//START_MODULE_HEADER////////////////////////////////////////////////////////
//
// File name:    fsyn_atom_dup.h
//
// Description: This operation duplicates nodes, and splits their fanout
//
//               Currently used for testing the API.
//
//               (1)  Node creation
//               (2)  Manipulation of oterms and iterms
//               (3)  Eventually will use timing information
//               (4)  Eventually will set preferred locations
//               (5)  Hosing the netlist.
//                       turn on "fsyn_hose_netlist_atom_dup=on" in quartus.ini
//                       this will remove certain connections during the duplication
   process
//
//               The current algorithm is
//               for each node
//                   if it is legal to duplicate (ie carry chain, global issues)
//                       duplicate the node
//                       copy the fanins
//                       split the fanouts for one oterm between the old and new node
//                       the other oterm, if it exists, is not copied or changed
//
//               [redacted]
//
// Authors:      Terry Borer
//
//               Copyright (c) Altera Corporation [redacted].
//               All rights reserved.
//
//
//END_MODULE_HEADER//////////////////////////////////////////////////////////

/* $Log:   X:/QUARTUS/FITTER/FSYN/FSYN_ATOM_DUP.H__  9
**
**    Rev 14.0.1.3   [redacted]   ihamer
**  SPR 105946
**  TO, [redacted]
**
**    Rev 14.0.1.2   [redacted]   ihamer
**  SPR 104650
**
**  TO, Tue [redacted]
**
**    Rev 14.0.1.1   [redacted]   ihamer
**  Latest duplication code
**  TO, [redacted]
**
**    Rev 14.0   [redacted]   max
**  Quartus II 2.2
**  SJ, [redacted]
**
**    Rev 1.4   [redacted]   ihamer
**  Modifications to duplication code
**  TO, [redacted]
**
**    Rev 1.3   [redacted]   ihamer
**  LC replication and improvements to api.
**  TO, [redacted]
**
**    Rev 1.2   [redacted]   ihamer
**  Adding faunction to do register packing
**  TO, [redacted]
```

```
**
**     Rev 1.1   [redacted]   tborer
**  A few new functions, a few new comments
**  TO, [redacted]
**
**     Rev 1.0   [redacted]   tborer
**  Initial Put
**  TO, [redacted]
*/
#ifndef INC_FSYN_ATOM_DUP_H
#define INC_FSYN_ATOM_DUP_H

// INCLUDE FILES ///////////////////////////////////////////////////////////////

// Include files in the following order below the
// corresponding headers.
//
// SYSTEM INCLUDE FILES
#include "fsyn_net_util.h"

// INTERFACE INCLUDE FILES FROM OUTSIDE MY SUB-SYSTEM

// INTERFACE INCLUDE FILES FROM WITHIN MY SUB-SYSTEM

// EXPORT INCLUDE FILES FROM WITHIN MY SUB-SYSTEM

// LOCAL INCLUDE FILES FROM WITHIN MY SUB-SYSTEM
// FORWARD REFERENCES FOR CLASSES /////////////////////////////////////////////
class FSYN_API;
// CLASS AND STRUCTURE DECLARATIONS ///////////////////////////////////////////

//START_CLASS_HEADER//////////////////////////////////////////////////////////
//
// Class name:  FSYN_ATOM_DUP
//
// Description: See the above file description
//
// Authors:     Terry Borer
//
//END_CLASS_HEADER////////////////////////////////////////////////////////////

class FSYN_ATOM_DUP : public FSYN_ALGORITHM_BASE
{
public:
    FSYN_ATOM_DUP
    (
        FSYN_API *fsyn_api,
        FSYN_ALGORITHM_PARAMETERS *params
    ) ;
    ~FSYN_ATOM_DUP(void);

    bool work (void);
    void init (int debug_level);

    virtual const char *get_name () { return ("FSYN_ATOM_DUP"); }

private:

    // these functions are used to randomly duplicate high fanout atoms
    void duplicate_high_fanout_nets();
    void duplicate_node_and_split_fanout([redacted]);
    void move_half_oterms_over_to_new_oterm([redacted]
[redacted]);
    bool can_duplicate_oterm([redacted]);
    bool can_duplicate_atom([redacted]);
```

D:\g71_jan15_b\quartus\fitter\fsyn\fsyn_atom_dup.h 3

```
    int m_debug_level;
    bool m_hose_netlist_for_testing;
};

//START_CLASS_HEADER/////////////////////////////////////////////////////////////
//
// Class name:  FSYN_DUP_OPERATION
//
// Description:
//
// Authors:     Ivan
//
//END_CLASS_HEADER///////////////////////////////////////////////////////////////
class FSYN_DUP_OPERATION
{
public:
    [redacted]

    int m_x;    // Destination prefered location
    int m_y;

    bool operator < (const FSYN_DUP_OPERATION& rhs) const
    {
        return [redacted];
    }

    bool is_equivalent (const FSYN_DUP_OPERATION& rhs) const
    {
        // dump ();
        // rhs.dump();
        return ((m_x == rhs.m_x) &&
                (m_y == rhs.m_y) &&
//              [redacted]
        );
    }

    FSYN_DUP_OPERATION
    (
        [redacted]
        int x, int y
        ) : [redacted]
            m_x (x), m_y(y)
    {
    }

    FSYN_DUP_OPERATION () :
        [redacted]
        m_x (-1), m_y(-1)
    {}

    void dump (void) const
    {
        FSYN_DEBUG.msg (2, "DUP OP: %s %x %d %d",
            [redacted]
            m_x,
            m_y
        );
    }
};
```

```
//START_CLASS_HEADER/////////////////////////////////////////////////////////////
//
// Class name:  FSYN_ATOM_DUP
//
// Description: Contains information on where a cell has duplicates.
//
// Authors:     Ivan
//
//END_CLASS_HEADER///////////////////////////////////////////////////////////////
class FSYN_DUP_MAP
{
public:
    class XY
    {
    public:
        int x, y;

        XY () : x(-1), y(-1) {}
        XY (int xx, int yy) : x(xx), y(yy) {}

        bool operator==(const XY& other) const
        {
            return (x==other.x && y==other.y);
        }
    };

    int     m_bin_id;

    struct LOC_BIN_PAIR
    {
        XY loc;
        int bin;
        bool original;
    };

    typedef STL_MAP(CDB_ATOM_NODE *, LOC_BIN_PAIR, less<CDB_ATOM_NODE *>) NODE_LOC_MAP;
    typedef NODE_LOC_MAP::iterator NODE_LOC_MAP_ITER;

    NODE_LOC_MAP m_map;

    FSYN_DUP_MAP () : m_map (), m_bin_id(0) {}

    CDB_ATOM_NODE *get_node_duplicate_at (CDB_ATOM_NODE *node, const XY &point, bool *
    original = NULL);
    void insert_node_duplicate (
        CDB_ATOM_NODE *new_node,
        CDB_ATOM_NODE *source_node,
        const XY &sink_loc,
        const XY &source_loc);
    void move_node (CDB_ATOM_NODE *node, const XY &loc);
    void dump ();
    void update_locations (FSYN_API *api);
};

//START_CLASS_HEADER/////////////////////////////////////////////////////////////
//
// Class name:  FSYN_ATOM_DUP
//
// Description: Replicates nodes on critical paths and tries to place them
//              together.
//
// Authors:     Ivan
//
//END_CLASS_HEADER///////////////////////////////////////////////////////////////
```

```
class FSYN_LOGIC_REPLICATION : public FSYN_ALGORITHM_BASE
{
private:
    [redacted]
    FSYN_NET_UTIL           m_net_util;
    FSYN_DUP_MAP            m_dup_map;
    CDB_VEC_OF_ATOM_NODE    m_atoms_to_add;
    CDB_VEC_OF_ATOM_NODE    m_atoms_to_delete;
    CDB_VEC_OF_ATOM_NODE    m_do_not_duplicate_list;
    int                     m_MAX_NUM_OP_PER_ITER;
    int                     m_LAB_OVERUSE_TRESHOLD;

    enum STATISTICS {
        LCS_DUPLICATED = 0,
        DUPLICATES_USED = 1,
        LCS_MOVED=2,
        LUTS_DUPLICATED=3,
        SKIPPED_SAME_LAB=4,
        SKIPPED_OTERM_NOT_COMB=5,
        SKIPPED_SOURCE_IN_A_CHAIN=6,
        SKIPPED_DRIVER_NOT_LC=7,
        SKIPPED_DRIVER_IN_QFBK=8,
        LAB_OVERUSE_REJECTION=9,
        SLACK_RATIO_GOOD_REJECTION=10,
        DUPLICATES_MERGED=11,

        STAT_ONE_PAST_LAST
    };

    static char *s_STAT_ARRAY_STRINGS[STAT_ONE_PAST_LAST];

    int m_stat_counts[STAT_ONE_PAST_LAST];

    int m_chip_labs_x;
    int m_chip_labs_y;
    int m_lc_count;

    struct LAB_FIELD
    {
        int num_original_lcs;
        int num_duplicated_lcs;
        int num_simple_registers;
        CDB_VEC_OF_ATOM_NODE simple_reg_vector;
        CDB_VEC_OF_ATOM_NODE node_vector;
    };

    typedef STL_VECTOR (LAB_FIELD) FSYN_LAB_VECTOR;
    typedef STL_VECTOR (FSYN_LAB_VECTOR) FSYN_LAB_MATRIX;

    FSYN_LAB_MATRIX m_lab_matrix;

    typedef STL_MULTISET(FSYN_DUP_OPERATION, less<FSYN_DUP_OPERATION >)
    FSYN_DUPLICATION_QUEUE;
    typedef FSYN_DUPLICATION_QUEUE::iterator FSYN_DUPLICATION_QUEUE_ITER;

    void initialize ();

    bool should_run_another_iteration ();

    bool perform_operation (const FSYN_DUP_OPERATION &op);
    void perform_operation_old (const FSYN_DUP_OPERATION &op);

    bool is_iterm_valid_for_duplication
    (
        [redacted]
```

```
            FSYN_DUP_OPERATION   *dup_op
        );

        void get_iterms_to_duplicate
        (
            FSYN_DUPLICATION_QUEUE *op_list
        );

        void duplicate_iterms
        (
            const FSYN_DUPLICATION_QUEUE &op_list
        );

        CDB_ATOM_NODE *create_lut_lc_copy
        (
            CDB_ATOM_NODE *source_node
        );

        void move_or_pack
        (
            CDB_ATOM_NODE *node,
            const FSYN_DUP_MAP::XY &source_loc,
            const FSYN_DUP_MAP::XY &sink_loc
        );

        bool are_all_fanouts_in_lab
        (
            [redacted]
            const FSYN_DUP_MAP::XY &sink_loc
        );

        int get_fanouts_at_dest_lab
        (
            [redacted]
            const FSYN_DUP_MAP::XY &sink_loc,
            [redacted]
        );

        void add_delete_and_pack_atoms (void);
        void clean_up_double_duplicates ();
        void initialize_chip_usage_array ();

        bool connect_lab_wide_signal
        (
            [redacted]
            const FSYN_DUP_OPERATION &op
        );
    public:
        FSYN_LOGIC_REPLICATION
        (
            FSYN_API *fsyn_api,
            FSYN_ALGORITHM_PARAMETERS *params
        ) ;
        ~FSYN_LOGIC_REPLICATION(void);

        bool work (void);
        bool work_new (void);

        virtual const char *get_name () { return "FSYN_LOGIC_REPLICATION"; }
    };


    #endif // INC_FSYN_ATOM_DUP_H
```